(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 806 731 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
12.11.1997 Bulletin 1997/46

(51) Int. Cl.[6]: G06F 17/30

(21) Application number: 97100673.9

(22) Date of filing: 17.01.1997

(84) Designated Contracting States:
DE FR GB NL SE

(30) Priority: 18.01.1996 US 588142

(71) Applicant: SUN MICROSYSTEMS, INC.
Mountain View, California 94043-1100 (US)

(72) Inventors:
- Bracho, Rafael
Cupertino, California 95014 (US)
- Sporkert, Tilman
San Jose, California 95131 (US)

(74) Representative: Kahler, Kurt, Dipl.-Ing.
Patentanwälte
Kahler, Käck, Fiener et col.,
Vorderer Anger 268
86899 Landsberg/Lech (DE)

(54) Database network

(57) A method and apparatus for publishing and receiving events to a network (120). A plurality of "publisher" entities publish information and a plurality of "subscriber" entities request and use the information. Publishers (102, 110, 116) and subscribers (104, 112, 118) are connected to each other through a network (120). The network (120) is a "store and forward" network whose routing is "content-based." The basic quanta of information is called an "event." Publishers (102, 110, 116) publish events and subscribers (104, 112, 118) subscribe to events that match criteria defined by the subscriber (104, 112, 118). Publication and subscription are performed asynchronously. Publishers (102, 110, 116) and subscribers (104, 112, 118) do not have direct knowledge of each other. The system (100) receives a published event from a publisher (102, 110, 116) and routes the event to all appropriate subscribers (104, 112, 118). Each subscriber (104, 112, 118) is guaranteed to receive all events published on the system (100) if, and only if, they match the subscription criteria specified by the subscriber. A legacy data base can be added to the network by way of a data base connector, which can be a publisher, a subscriber, or both.

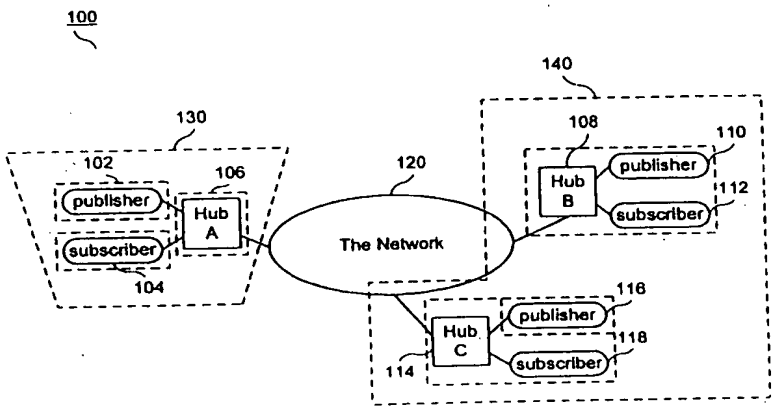


FIG. 1

EP 0 806 731 A2

ing systems and running various types of applications. The described embodiment utilizes a distributed-object environment and a preferred embodiment of the present invention implements the Common Object Request Broker (CORBA) standard.

In accordance with the purpose of the invention, as embodied and broadly described herein, the invention relates to a method of connecting a data base to a publisher/subscriber network, so that the data base can publish events to the network, the method comprising the steps, performed by the data processing system, of: providing a link so that the data base can communicate with a data base connector computer program by way of a transaction monitor computer program; setting up the data base connector as a publisher hub in the network; receiving input, by the data base, from a user that alters data in the data base; informing the transaction monitor that the data has been altered; receiving input from the user indicating that the altered data should be committed; informing the transaction monitor that the altered data is committed; and sending, by the data base connector, in accordance with a message from the transaction monitor that the data is committed, a published event to the network in accordance with the altered data.

Objects and advantages of the invention will be set forth in part in the description which follows and in part will be obvious from the description or may be learned by practice of the invention. The objects and advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims and equivalents.

Brief Description of the Drawings

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate several embodiments of the invention and, together with the description, serve to explain the principles of the invention.

Fig. 1 is a block diagram of a networked computer system in accordance with a preferred embodiment of the present invention.

Fig. 2 is a flow chart showing steps performed to install an application, such as a publisher, on a hub of Fig. 1.

Fig. 3 is a flow chart showing steps performed by a publisher of Fig. 1 to publish events.

Fig. 4 is a flow chart showing steps performed by a subscriber of Fig. 1 to subscribe to events.

Fig. 5 is a block diagram showing details of a hub of Fig. 1.

Fig. 6(a) is a diagram showing data structures stored in a memory of the hub of Fig. 5.

Fig. 6(b) is a listing of details of the data structures of Fig. 6(a).

Fig. 7 is a diagram of additional data structures stored in the memory of the hub of Fig. 5 for the purpose of storing information about clients of the hub.

Fig. 8 shows a format of an envelope data structure of an event.

Fig. 9 shows a format of a routing block of an event.

Fig. 10 is a flow chart showing details of steps performed by the hubs of Fig. 1 to populate the data structures of Fig. 6(a).

Fig. 11 is a diagram showing examples of the data structures of Fig. 6(a) populated with data.

Fig. 12 is a flow chart showing details of steps performed by the hubs of Fig. 1 to send published events to subscribers.

Fig. 13 is a block diagram of a data base application incorporated into the network of Fig. 1.

Fig. 14 is a flow chart showing steps performed when the data base is a subscriber.

Fig. 15 is a flow chart showing steps performed when the data base is a publisher

Detailed Description of the Preferred Embodiments

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

I. Overview

Fig. 1 is a block diagram of a networked computer system 100 in accordance with a preferred embodiment of the present invention. Networked computer system 100 includes a plurality of publishers 102, 110, and 116 and a plurality of subscribers 104, 112, and 118. Publisher 102 and subscriber 104 are connected to a network 120 through a hub 106. Publisher 110 and subscriber 112 are connected to network 120 through a hub 108. Publisher 116 and subscriber 118 are connected to network 120 through a hub 114. Hub 106 and its connected publishers and subscribers are in a first territory 130. Hubs 108 and 114 and their connected publishers and subscribers are in a second territory 140. Other territories (not shown) also exist in network 120. "Hubs," "publishers," "subscribers," and "territories" are discussed below in turn.

As indicated by dotted lines in Fig. 1, each publisher, subscriber, and hub can be located in separate computers

```
//filename: order:ddl

interface SalesOrder {
struct Address {
                        string street;
                        string city;
                        string state;
                        unsigned long zip;
        };

        attribute string customer_name;
        attribute Address customer_address;

        };
```

The above syntax defines an event as an "interface" and defines the data members in the event as "attributes."

After one or more events have been defined, the events are translated into structure definitions of the C programming language. Preferably, this translation is performed by an OMG IDL compiler, which generates a header file and a code file for each event. The generated header file contains simple C-language structures that represent the event declarations for the typed functions defined in the code file. The C structure definition generated for the above example event of type SalesOrder is:

```
typedef struct SalesOrder_Address {
        nxsString street;
        nxsString city;
        nxsString state;
        nxsULong zip;
} SalesOrder_Address;

typedef struct SalesOrder {
        nxsString customer_name;
        SalesOrder_address customer_address;
} SalesOrder;
```

A person of ordinary skill in the art will easily understand how the IDL of the example translates into the C programming language structure definitions shown above.

The code file generated from the event definition contains procedures for manipulating events of the defined type (here, for the SalesOrder event). The procedures perform at least the following operations:

For use by a publisher:

Create event of type SalesOrder.

Publish event of type SalesOrder.

Destroy event of type SalesOrder

that a publisher could execute and send those events. This information is needed to build routes to subscribers, as described below. Although not shown in the Figure, other publishers connected to the same hub may operate asynchronously with the steps of Fig. 3.

Subscribers subscribe to events of particular event types. Fig. 4 is a flow chart showing steps performed by a subscriber of Fig. 1 during operation of the subscriber. The subscriber first registers itself as a client of the hub. Then for each subscription, it registers a "call-back" procedure.

As shown in step 402, the subscriber next registers itself as a client with the hub to which it is connected. The subscriber then registers a "subscription" for the event type that it wishes to receive through the hub. (The subscriber can look at the hub to see what types of events are advertised.) A subscription specifies a type of event, and can further specify a "filter" indicating that the subscriber wishes to receive only events of a certain type that also have certain values in certain fields.

In step 404, for each subscription, the publisher defines a predetermined "call-back" procedure. The call-back procedure will be initiated by the hub whenever the hub determines that the subscriber has received an event of a certain type. The format of the callback may be different for each operating system on which the present invention is implemented and the call-back procedure can be practically any procedure that is performed when an event is received. For example, in step 406 of Fig. 4, the call-back procedure could print the contents of the event (using one of the event manipulation procedures designed to print an event of that type). Because a call-back procedure is defined for each subscription, a call-back procedure must be defined for each type of event to which the subscriber plans to subscribe. In the described embodiment, an event passed into a call-back procedure is destroyed when the call-back procedure returns. Thus, if the subscriber wants to keep any information of the event, the information should be copied out of the event within the call-back procedure.

The subscriber loops until it is time to cease execution (see step 408). The call-back procedure may be activated zero, one, or many times during this loop, as events of the specified type and values are received by the hub. If an event subscribed to by the subscriber is received, the call-back procedures is executed in step 406. If, in step 408, the subscriber determines that it is time for it to quit, the subscriber optionally unregisters its subscription, unregisters itself with the hub, and ceases execution. Subscriptions can also be retained in the hub, to receive future events while disconnected. The hub will queue events arriving for the subscriptions. Although not shown in the Figure, other subscribers connected to the same hub may perform steps asynchronously with the steps of Fig. 4.

The following examples are written in the C programming
language and show example software for a publisher and subscriber.

```
        return (0);
}

/*
* subscribe.c
*/

#include <nexus.h>
#include "nxs_output/order_nxs.h"

/* Callback function to be called when each event is received */
void event_callback (
        nxsClientID id,
        nxsSubscriptionID sub_id,
        nsxULong seq_num_high,
        nsxULong seq_num_low,
        SalesOrder *the_event,
        void *client_data)
{

        char *st;

        printf("Event received!\n");
        st = nxsStringify_SalesOrder(the_event);
        printf(st);
        free(st);

}

int main(int argc, char **argv)
{

        Sales Order *event;
        nxsClient ID id;
        nxsSubscriptionID sub_id;
        /* Register Client and subscription */
        /* (Only subscribe to specific ZIP codes) */
        nxsCreateClient ("order subscriber",NULL,NULL, 0,&id);
        nxsRegisterSubscription SalesOrder(id,
                "customer_address.zip == 95000",
                event_callback,NULL,&sub_id);
```

of events and only pass events to the subscribers that match certain criteria. A filter preferably is specified as a expression string sent by the subscriber as a parameter to the routine for registering subscriptions. The expression string syntax for a content filter in the described embodiment is as follows. (Of course, any appropriate syntax could be used):

symbol meaning		types allowed for
=	equal to	all basic types
!=	not equal to	all basic types
>	greater than	numeric and string types
>=, <=	greater than or equal	numeric and string types
and	logical AND	expressions or Booleans
or	logical OR	expressions or Booleans
not	logical NOT	expressions or Booleans

Event attribute names are used to denote which field in the event should be compared. Sub-fields (those inside structures) are specified using dot notation. Names can also be enumerated types, as is known to persons familiar with the C programming language. Once a content filter has been specified, it is used during event routing as described below in connection with Figs. 8-12. Information describing each content filter for a subscription is stored in field 774 of Fig. 7.

Fig. 6(a) is a diagram showing data structures stored in a memory 690 of hub 106 of Fig. 5. All hubs contain similar data structures. The described implementation uses a distributed objects model, but any appropriate organizational structure could be used. Details of the data structures of Fig. 6(a) are listed in Fig. 6(b). Ideally, the data structures of Fig. 6(a) describe all advertisements in the system and indicates a neighbor hub to which the current hub should send events of certain types. Figs. 10 and 12, respectively, describe the steps to populate the data structures of Fig. 6(a) and to use to the data structures of Fig. 6(a) to route events among the hubs. Fig. 11 shows an example of the data structures populated with data.

Hub 106 (the "current hub") includes data objects representing: each client of the current hub that is a subscriber (indicated by "Client" 684 and "S" 695) and each neighboring hub connected to the current hub ("Neighbor A" 696 and "Neighbor B" 697). The current hub keeps track of the subscription objects of its neighbor hubs. Subscription objects "S" 650 represent all subscribers that can be reached through the respective neighbors. Each neighbor object has two associated cost values: "mycost" and "theircost". "Mycost" represents a cost of sending information from the current hub to the neighbor. "Theircost" represents a cost of sending information from the neighbor to the current hub. "Theircost" used to define a route, as described below. The subscribing hub will ask the "publishing" hub with the lowest "theircost" to forward the events. "Mycost" preferably is only used to inform the neighbors of how much it would cost to the current hub to send information through that link. A "cost" may be figured in a number of ways. For example, a cost may represent a financial cost, a cost in units of time, a cost as a measure of convenience, or any other appropriate cost.

Hub 106 contains a ("RemoteAd") object for each advertisement that has been registered in the system (i.e., for each intent to publish, step 202 of Fig. 2). Each RemoteAd object points to one or more "AdSource" objects, each of which represents a path between the current hub and the hub on which the publisher of the ad resides. Each AdSource object stores a cost associated with its path to the original publisher of the ad. Thus, the "cost" value for each AdSource contains the total cost for sending an event from its source to the neighbor of the current hub or, alternately, to the current hub.

Each AdSource object has an associated list of "sink objects." (SinkCa and sinkNb preferably are located in a single list, although not shown that way in the Figure.) Each sink object has an associated list of subscriptions. For example, SinkCa has a list 505 of all subscriptions of Client 694 that matched to the advertisement of RemoteAd 510. Similarly, SinkNb has a list 515 of all subscriptions of Neighbor B (and of all subscriptions that can be reached through Neighbor B) that have matched the advertisement of RemoteAd 510.

Each neighbor object also has an associated AdSourceRef list that points to AdSources for all paths between a publisher and the neighbor. The paths are used for routing system events between hubs, as described below.

IV. Event Routing Between Hubs

Fig. 10 is a flow chart showing details of steps performed by hubs 106, 108, and 114 of Fig. 1 to populate the data structures of Fig. 6(a). These steps are performed, for example, when the network is first initialized. Subsequently, var-

(step 1203). The steps of Fig. 12 are performed by various hubs as each event is routed through the system. In the following paragraphs, the hub performing the steps of Fig. 12 is termed the "current hub."

If the current hub receives an event from one of its connected publishers (step 1202), the preprocessor 514 of the current hub initially adds an "envelope" to the event (step 1204). A format of an event envelope is described below in connection with Fig. 8. Preprocessor 514 then determines in step 1204 whether the event type of the event is registered in the hub. If not, the publisher cannot publish the event. If so, in step 1206, the hub places the event in the hub's incoming event queue for further processing. In the described embodiment, the event queue discards duplicate events (as determined by a sequence number or time stamp of the event envelope).

If the current hub receives the event from a neighboring hub (step 1203), the neighboring hub places the event in the incoming event queue of the current hub.

In step 1208, filter 518 of the current hub gets the event from the current hub's incoming event queue. Filter 518 then increments the time-spent field of the envelope, which indicates an amount of time spent in queue. In step 1210, filter 518 adds a routing block to the event. Each hub that routes the event adds another routing block to the event. A format of two routing blocks is shown in Fig. 9.

In step 1212, the current hub locates a "current" AdSource object that represents all the subscribers which are interested in the event coming from that particular AdSource. This will usually represent a least a cost path between the current hub and the subscribers, (note that the described embodiment does not reroute once an AdSource data structure is established. The AdSource object is located by either searching all RemoteAds for the current hub (if the event came from a publisher attached to the current hub) or from searching the AdSourceRef list for the neighboring hub (if the event came from a neighboring hub). Steps 1216-1222 send the event to one or more subscribers. Steps 1216-1222 are performed for each sink object attached to the AdSource. Thus, steps 1216-1222 are performed for each subscriber that has asked to receive events of the type being routed.

If, in step 1216, the time-to-live for the event is less than the time-spent field of the envelope, then the event has expired and is not routed further by the current hub (step 1218). Otherwise, in step 1220, filter 518 determines whether the event has contents that fall within parameters specified by the subscriber. The subscriber specified the content filter at the time it subscribed to the event type (see field 774 of Fig. 7). For example, a subscriber may have requested to receive only those SalesEvents having where the customer lives in Los Angeles. If the event has values in the range specified by the subscriber, then the event is sent to the subscriber in step 1222 (either directly or by way of its hub, as described below). Each subscriber may have specified a different content filter (or no content filter). For example, a first subscriber may have specified that it wishes to receive SalesEvents where the customer lives in Los Angeles and a second subscriber may have specified that it wishes to receive SalesEvents where the customer lives in San Francisco. In this example, it is possible that the event will match the first subscriber but not the second subscriber.

In step 1222, filter 518 sends the event to the matching subscriber. The subscriber may be a client of the current hub, in which case the event is sent directly to the subscriber. Alternately, the subscriber may be connected to a neighboring hub, either directly or indirectly. If a matching subscriber is connected to a neighboring hub, the current hub will route the event to the neighboring hub (unless the neighboring hub is the hub originating the event, as determined from field 802 of the envelope, see Fig. 8). The current hub also will not route the event to a hub that has already seen the event (as determined from the routing blocks attached to the event, see Fig. 9).

In the described embodiment, the current hub only routes the event to a neighboring hub a single time, even if the neighboring hub has more than one matching subscription. Thus, if subscription object 651 of Fig. 6(a) matches the event, the event is forwarded to neighboring hub B. If subscription object 652 also matches the event, it is not necessary to forward the event to the neighboring hub B a second time. When neighboring hub B receives the event, it also will perform the steps of Fig. 12 to determine whether any of its client subscribers match the event.

Fig. 8 shows a format of an envelope data structure of an event. The preprocessor of Fig. 5 adds an envelope to an event received from its subscriber before the event is published. An envelope is associated with each event and includes an origin hub 802, an adname 804, a publisher name 806, a territory name 808, an initial time stamp 810, a time to live 812, a priority 814, flags 816, and a time spent field 822. Flags 816 are unused in the described embodiment. In an alternate embodiment, the flags may indicate, e.g., whether an event is "persistent" or "transient."

Fig. 9 shows an example format of two routing blocks of an event. Each hub adds a routing block to an event that it routes through itself. The API also includes a get_current_route_info routine that returns information about the actual route that an event took to reach a certain subscriber. Routing information is stored by the system as linked list, one entry for each hub visited. Each entry includes: an outgoing time stamp for the current hub, a hub name of the current hub, and a territory name of the current hub. The routing information optionally includes a sixty-four bit sequence number initially assigned to the event by the publisher.

V. Territories

Each territory 130 and 140 of Fig. 1 is a collection of hubs with a common event dictionary. Territories reflect organization, and are not necessarily tied to geography or direct physical connectivity. In the described embodiment, all hubs

If, in step 1507, the user (or software making changes) indicates that the transaction should commit, this fact is communicated to transaction monitor 1306 in step 1510. Transaction monitor 1306 informs data base connector 1310, which pulls the committed data from data base 1308 and publishes an event including the committed data. Events are published as shown in Fig. 12. If, in step 1512 the user (or software making changes) indicates that the transaction should roll back, this fact is communicated to transaction monitor 1306. Transaction monitor 1306 informs data base connector 1310, which "forgets" about the changes of step 1506 and does not publish an event. Thus, the described embodiment takes into account the commitment and roll back operations common to commercial data base operation.

In an alternate embodiment, where the data base product 1302 used does not include a transaction monitor, data base connector 1310 can act as a transaction monitor when connecting with data base 1308. Data base connector 1310 still acts as a publisher and/or subscriber in this configuration. Data base 1308 thinks that it is connected to a transaction monitor, when it is really connected to data base connector 1310.

VII. Summary

In summary, a preferred embodiment of the present invention includes a plurality of "publisher" entities, who publish information, and a plurality of "subscriber" entities, who request and use the information. Publishers and subscribers are connected to each other through a network. The network is a "store and forward" network whose routing is "content-based."

In the described embodiment of the present invention, the basic quanta of information is called an "event." Publishers publish events and subscribers subscribe to events that match criteria defined by the subscriber. Publication and subscription are performed asynchronously. Publishers and subscribers do not have direct knowledge of each other. The system receives published event from a publisher and routes the event to all appropriate subscribers. Each subscriber is guaranteed to receive all events published on the system if, and only if, they match the subscription criteria specified by the subscriber.

The described embodiment of the present invention makes it easy to integrate legacy systems, legacy applications, and legacy hardware into the system and attempts to minimize the amount of information that a user must learn to use the system. A legacy data base can be added to the network by way of a data base connector, which can be a publisher, a subscriber, or both.

Other embodiments will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope of the invention being indicated by the following claims.

APPENDIX A

Object references are transfered in stringified form.

System events are used for several styles of communication. During connection establishment, system events are used as peer-to-peer messages. These events are not seen by other hubs. Most other events affect the state of the territory and may be forwarded throughout the territory graph. A few events may be sent from a hub to itself.

NXS_SYS_EVENT_CONNECT_HUB

```
/string/      Sending hub name
/string/      Sending hub territory
string        Sending hub incoming EventQueue::PushConsumer
ulong         Receiving hub's cost
ulong         Sending hub's cost
```

This event is sent as a result of a call to NexusAdmin::HubAdmin::connectHub. The sending hub has already created the Neighbor and EventQueue for the connection. The receiving hub does likewise when it receives the system event. Note that the sending hub already has the incoming EventQueue::PushConsumer of the receiving via an argument to NexusAdmin::HubAdmin::connectHub. If this is the first connection for the receiving hub (it is joining a territory), then it needs to get the territory's shared data (event types, etc.). System events are used to request this information from the sending hub: NXS_SYS_EVENT_REQUEST_EVENTTYPES. and NXS_SYS_EVENT_REQUEST_ADVERTISEMENTS. If the hub is already a member of the territory, it sends its routes to the sender with NXS_SYS_EVENT_NEW_ROUTES. The receiving hub always requests routes from the sender with NXS_SYS_EVENT_REQUEST_ROUTES. If the connection cannot be created, for example the territory name is wrong, the receiver sends a NXS_SYS_EVENT_CONNECT_FAILED.

NXS_SYS_EVENT_CONNECT_FAILED

```
/string/      Sending hub name
/string/      Sending hub territory
/string/      Reason for connection failure
```

Sent when connection establishment failed. Clean up data structures.

NXS_SYS_EVENT_DISCONNECT_HUB

```
/string/      Sending hub name (or neighbor name)
/string/      Sending hub territory
```

This system event can be sent to self or to another hub. Sent to self as a result of call to NexusAdmin::HubAdmin::disconnectHub. In this case, the first argument is the name of the neighbor hub. The hub sends a similar event to the neighbor (containing the name of the sending hub) and does a connection shutdown. This may cause the following system events to be sent: NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS. NXS_SYS_EVENT_DELETE_ROUTES. and NXS_SYS_EVENT_DELETE_ADVERTISEMENTS. One the connection clean-up events are sent, a NXS_SYS_EVENT_END_OF_STREAM is delivered.

A-2

```
                    Repeated A times:
/string/            Advertisement name
/string/            Event type name
long                Priority
string              Storage mode; "p" or "t" for persistent or tran-
                    sient
ulong               Time to live (seconds)
/string/            Originating hub name
/string/            Originating hub territory
```

Create RemoteAds for the listed advertisements. The originating hub is where the advertisement was created. The system event is forwarded to all connected neighbors excepted the sender and neighbors who have already seen the event.

NXS_SYS_EVENT_CONNECTION_READY

```
/string/            Sending hub name
/string/            Sending hub territory
```

The sending hub is ready to use the connection. If the receiver is also ready, it replies with a NXS_SYS_EVENT_CONNECTION_READY. If the connection is already established the system event is ignored.

NXS_SYS_EVENT_FORWARD_ADVERTISEMENT

```
string              Advertisement name
```

Send the named advertisement to all connected neighbors. Only sent to self when advertisements are created.

NXS_SYS_EVENT_CHANGE_ADVERTISEMENT

```
/string/            Advertisement name
/string/            Originating hub name
/string/            Originating hub territory
long                Priority
string              Storage mode; "p" or "t" for persistent or tran-
                    sient
ulong               Time to live
```

Update the matching RemoteAd with the given values. The system event is forwarded to all connected neighbors except the sender and hubs that have already seen the event.

NXS_SYS_EVENT_DELETE_ADVERTISEMENTS

```
long                Number of Advertisements A
                    Repeated A times:
/string/            Advertisement name
/string/            Originating hub name
/string/            Originating hub territory
```

A-4

The System Events

```
/string/      Advertisement name
/string/      Originating hub name
/string/      Originating hub territory
```

Delete the routes (AdSource) for the given advertisements (RemoteAd). Any subscriptions which were fed by that route are failed. This may result in NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS sent to neighbors who have subscriptions registered for the advertisement. Forwarding of each deleted route is considered separately and only if the last route to the advertisement was deleted. The system event is not forwarded to the sender or the originating hub.

NXS_SYS_EVENT_CHANGE_CONNECTION

```
/string/      Sending hub name
/string/      Sending hub territory
long          Delta to sending hub's connection cost
```

Update their cost for the connection.

NXS_SYS_EVENT_NEW_SUBSCRIPTIONS

```
long          Number of subscriptions S
              Repeated S times:
/string/      Advertisement name
/string/      Advertisement hub name
/string/      Advertisement hub territory
/string/      Filter expression
ulong         Owner id in sender
ulong         Subscription id in sender
```

Register subscriptions from the sender against the given advertisements. Create a Subscription on the sending neighbor and tell the RemoteAd about it. The RemoteAd tells the current AdSource which creates a Sink for the Neighbor (if one does not yet exist). The Sink adds a SubscriptionRef for the Subscription.

If the advertisement is from another hub, forward a system event for the new subscription to the neighbor supplying the current route.

The owner and subscription ids are used to locate the subscription when it is canceled and to identify it when it fails.

If there is a failure during subscription registration, an NXS_SYS_EVENT_FAIL_SUBSCRIPTIONS is returned to the sender.

NXS_SYS_EVENT_CANCEL_SUBSCRIPTIONS

```
long          Number of subscriptions S
              Repeated S times:
ulong         Owner id in sender
ulong         Subscription id in sender
```

A-6

the network, the method comprising the steps, performed by the data processing system, of:

providing a link so that the data base can communicate with a data base connector computer program by way of a transaction monitor computer program;
setting up the data base connector as a publisher hub in the network;
receiving input, by the data base, from a user that alters data in the data base;
informing the transaction monitor that the data has been altered;
receiving input from the user indicating that the altered data should be committed;
informing the transaction monitor that the altered data is committed; and
sending, by the data base connector, in accordance with a message from the transaction monitor that the data is committed, a published event to the network in accordance with the altered data.

2. The method of claim 1, further including the steps of:

receiving input from the user indicating that the altered data should be rolled back;
informing the transaction monitor that the altered data is rolled back; and
refraining, by the data base connector, from sending the altered data to the network in accordance with the roll-back.

3. A method of connecting a data base to a publisher/subscriber network, so that the data base can subscribe to events to the network, the method comprising the steps, performed by the data processing system, of:

providing a link so that a data base connector computer program can communicate with the data base;
setting up the data base connector as a subscriber hub in the network;
receiving, by the data base connector, an event from the network;
sending, by the data base connector, data in accordance with the received event to the data base; and
committing, by the data base connector, the data in the data base.

4. The method of claim 1, further comprising the step of:

sending the event, by the data base connector, to one of its neighbor hubs, where a data structure in the data base connector indicates that the neighbor hub is on the least-cost path to a subscriber for the event.

5. The method of claim 1, further comprising the steps of:

receiving an event by the data base connector;
determining, by the data base connector, in accordance with a data structure of the data base connector, that the data base has subscribed to the event; and
sending, by the data base connector, the event to the data base.

6. The method of claim 1, further comprising the step of:

receiving an event, by the data base connector;
determining, by the data base connector, in accordance with a data structure of the data base connector, that a neighbor hub is connected either directly or indirectly on the least-cost path to a subscriber for the event; and
sending, by the data base connector, the event to its neighbor hub, so that the event can be forwarded to the subscriber.

7. The method of claim 1, wherein the sending step includes the step of:

sending an event, by the data base connector, to one of its neighbor hubs when a data structure in the data base connector indicates that the data base subscribes to events having the type of the event.

8. The method of claim 1, wherein the sending step includes the step of:

sending an event, by the data base connector, to one of its neighbor hubs when a content filter in a data structure of the data base connector indicates that the data base subscribes to events having values found in the event.

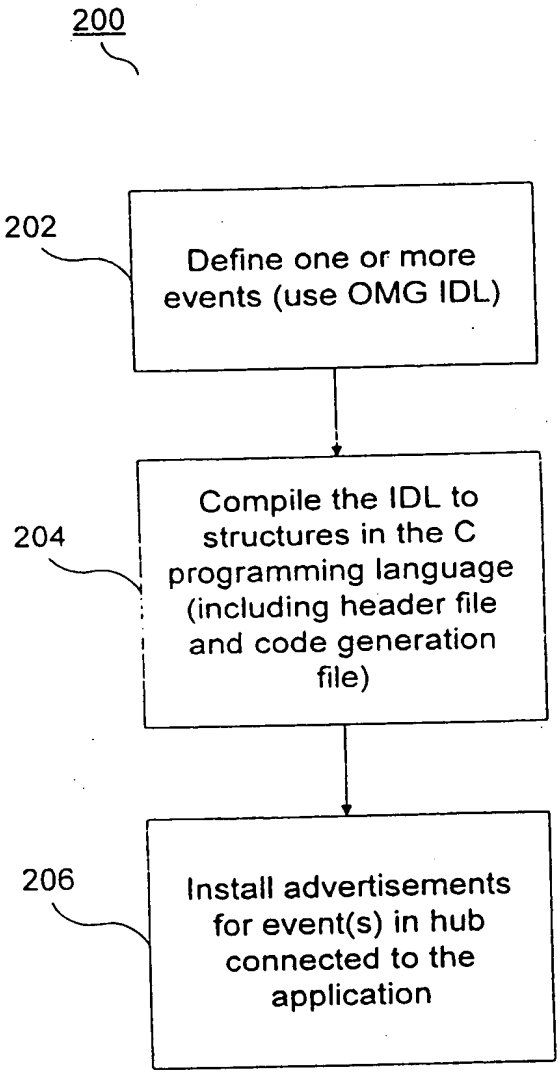
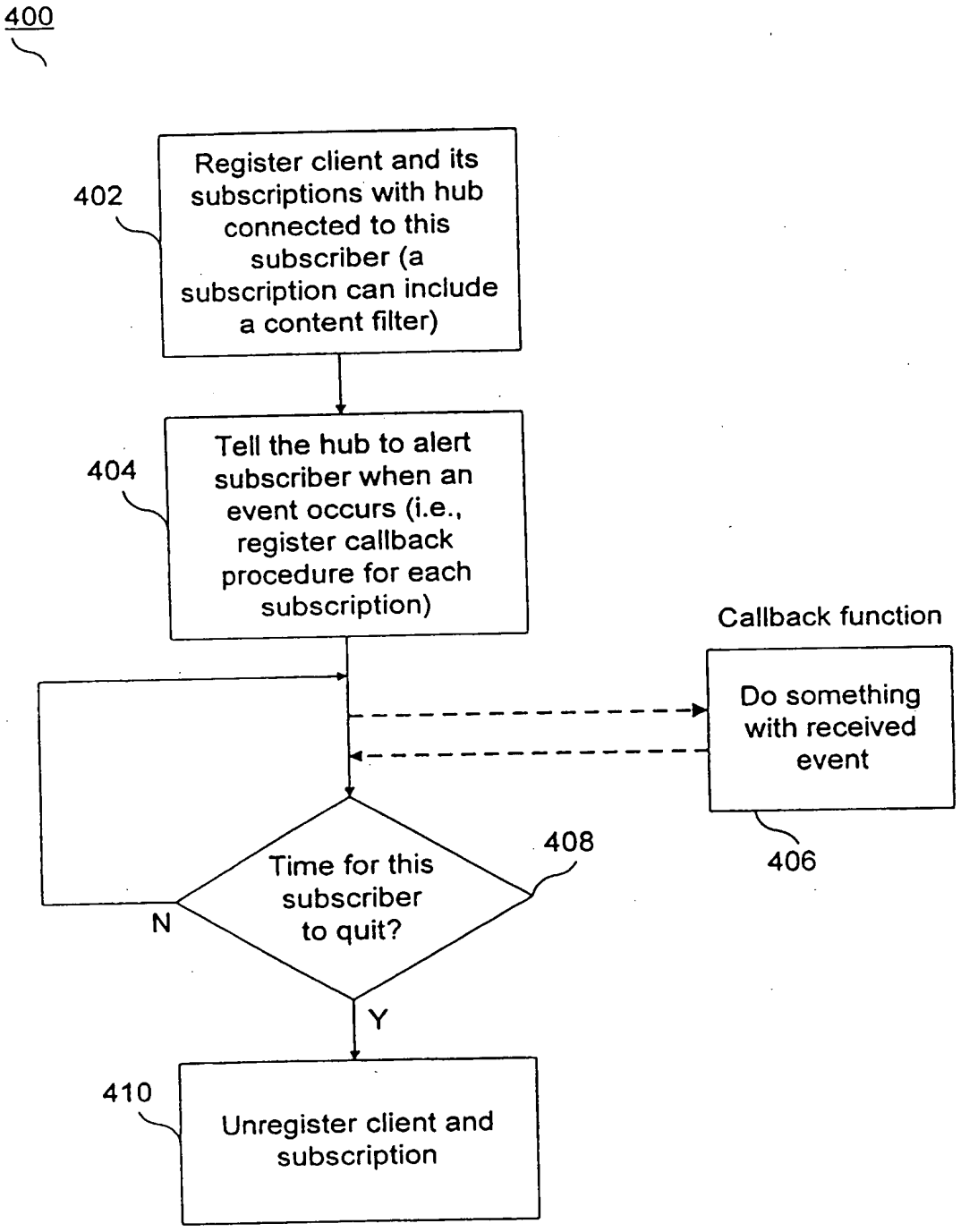
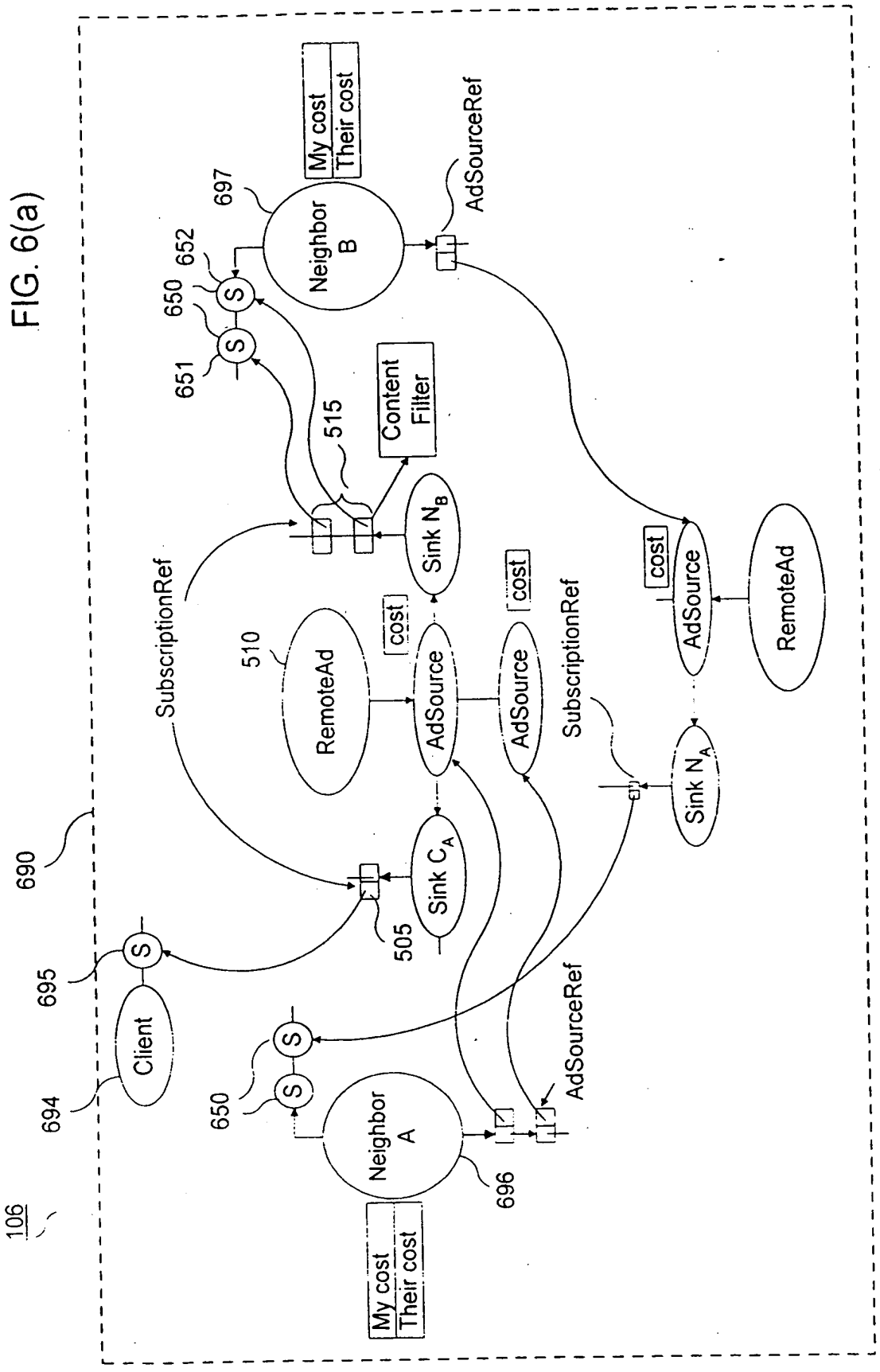


FIG. 2
Installing an Application (Publisher) on a Hub



Subscribing to Events
FIG. 4

FIG. 6(a)


106
690
694
Client
695
S
505
Sink C_A
510
RemoteAd
cost
AdSource
SubscriptionRef
650
651
652
S
515
Content Filter
Sink N_B
697
Neighbor B
My cost
Their cost
AdSourceRef
696
Neighbor A
My cost
Their cost
AdSourceRef
Sink N_A
SubscriptionRef
AdSource
cost
RemoteAd

Installed advertisements for this hub

700

Advert Name	Event Description	Freq of Publication	Event Type	Priority	Time To Live

Registered clients of this hub

734 736 737 740 742

Application Name	User Name	User Password	Hub Name	Territory Name	Flags	Client ID

730

Registered publications for this hub

764

760

Client ID	Advert Name	Publication ID

Registered subscriptions for this hub

774

770

Client ID	Content Filter Info (if any)	Event Type	Subscription ID

FIG. 7

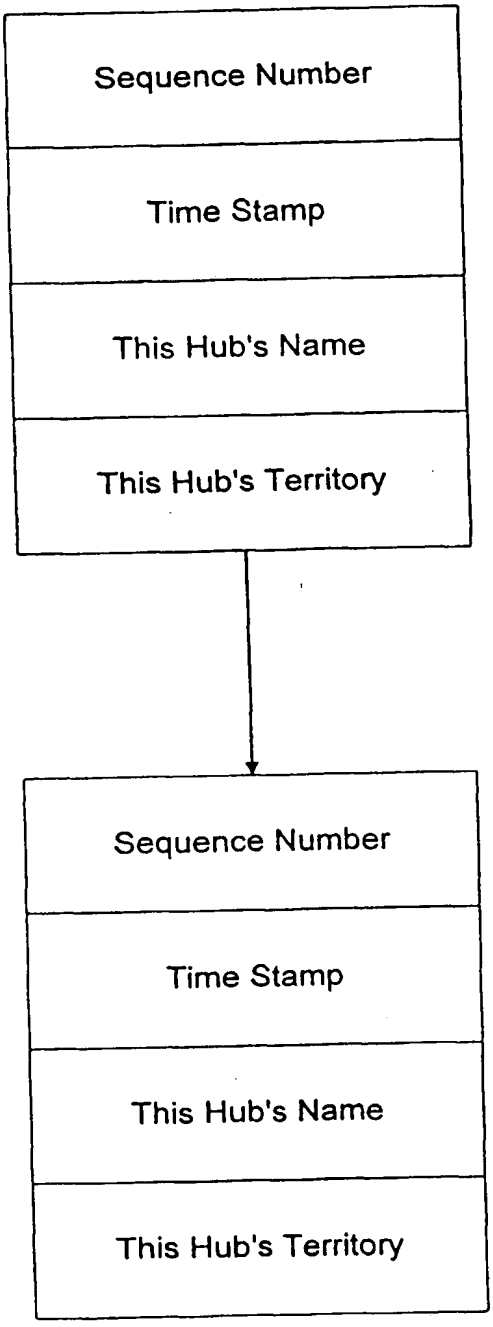


FIG. 9
Routing Blocks of an Event

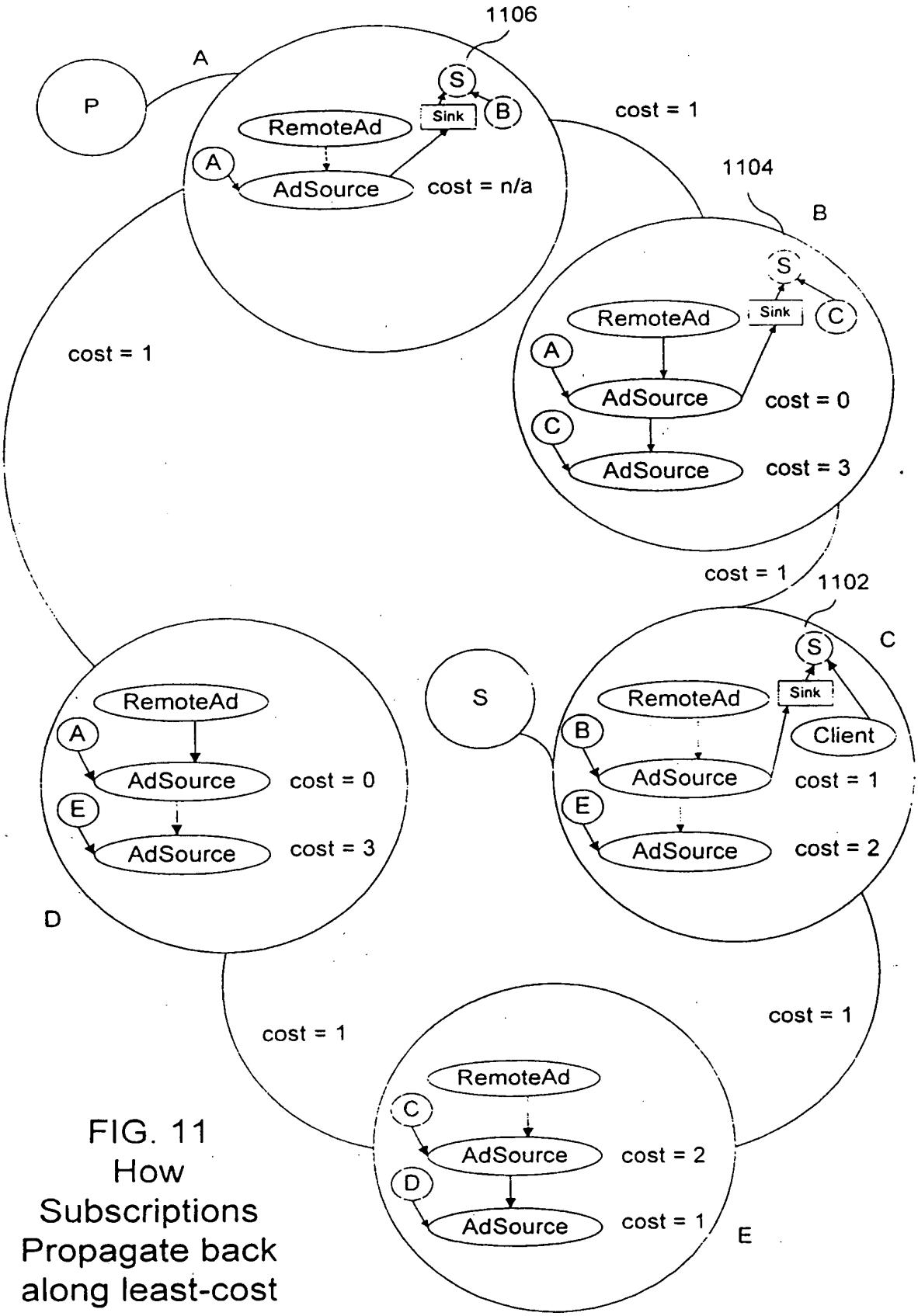


FIG. 11
How Subscriptions Propagate back along least-cost route

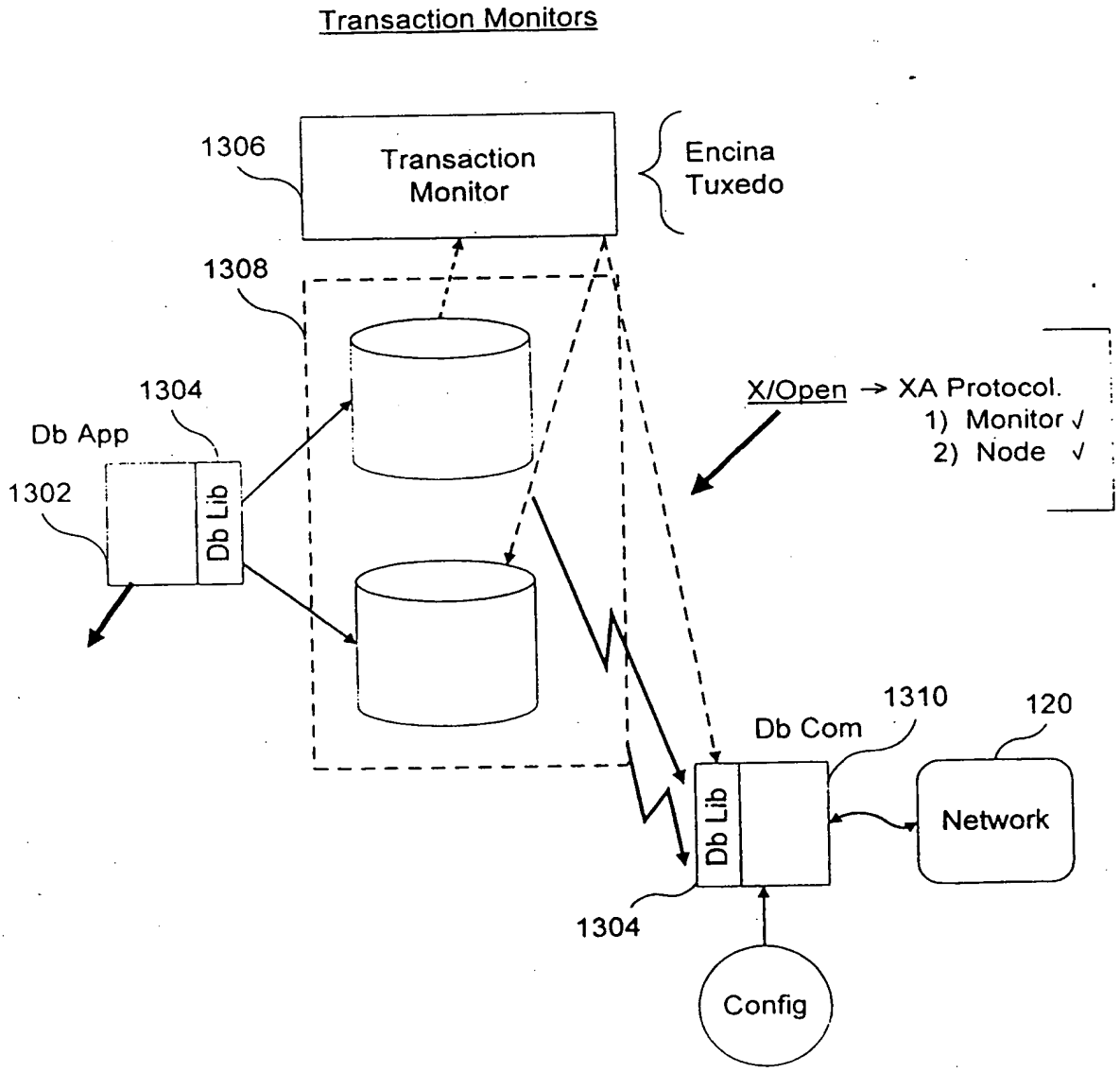

Transaction Monitors
1306
Transaction Monitor
Encina Tuxedo
1308
1304
Db App
1302
Db Lib
X/Open → XA Protocol.
1) Monitor √
2) Node √
1310
120
Db Com
Db Lib
Network
1304
Config

FIG. 13

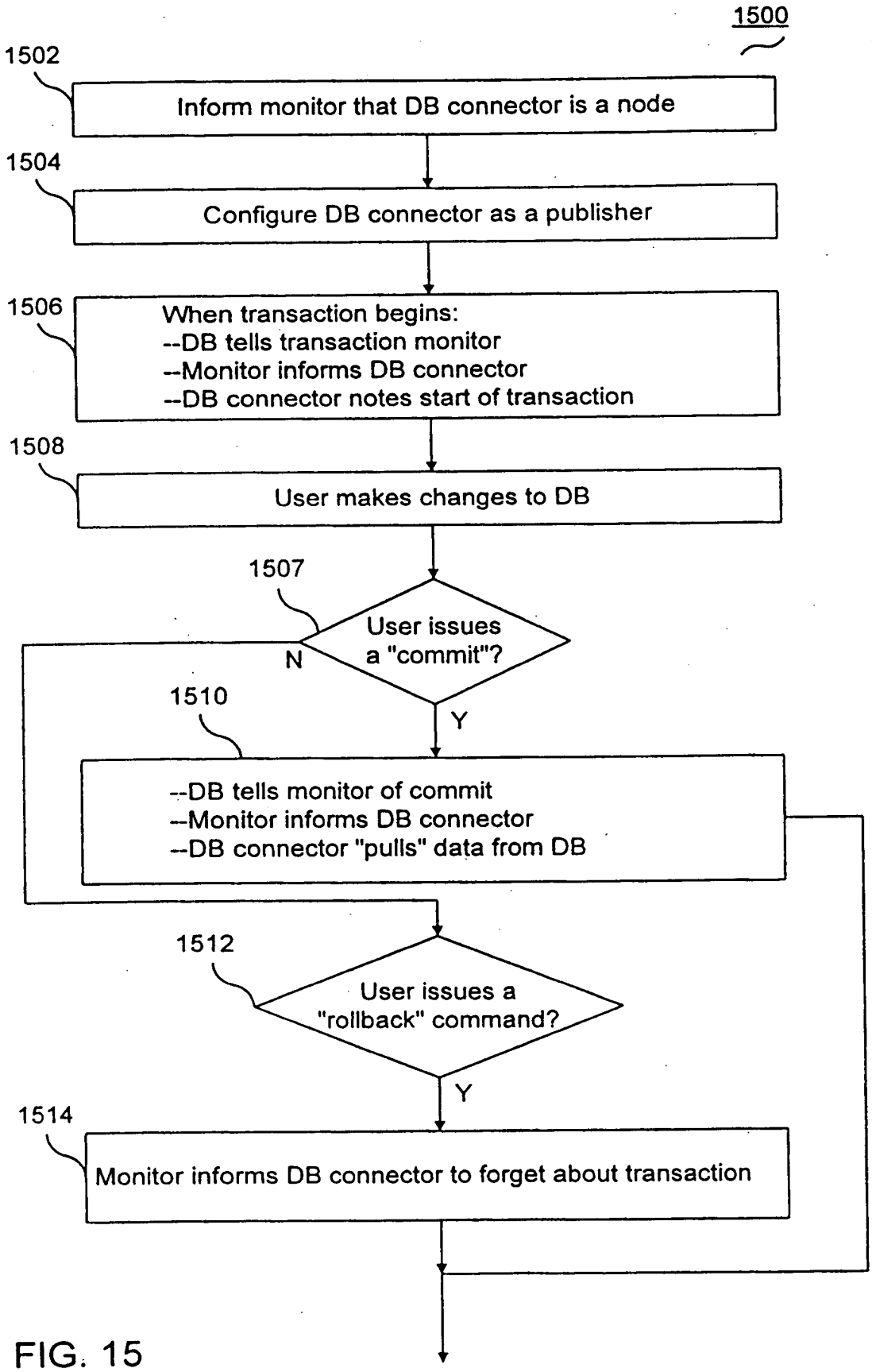

1500
1502
Inform monitor that DB connector is a node
1504
Configure DB connector as a publisher
1506
When transaction begins:
--DB tells transaction monitor
--Monitor informs DB connector
--DB connector notes start of transaction
1508
User makes changes to DB
1507
User issues a "commit"?
N
Y
1510
--DB tells monitor of commit
--Monitor informs DB connector
--DB connector "pulls" data from DB
1512
User issues a "rollback" command?
Y
1514
Monitor informs DB connector to forget about transaction

FIG. 15